

# **IZMIR UNIVERSITY OF ECONOMICS**

## **SE 115 – INTRODUCTION TO PROGRAMMING I**

**2024-2025 FALL SEMESTER**

### **SE 115 MAPS PROJECT**

**NAME:** Burcu Tanrıverdi

**STUDENT ID:** 20230202056

#### **ABSTRACT**

The purpose of this project is to read the text file and ensure it is the desired format and apply the algorithm that finds the shortest path if there are any and write it into the output file. Project only contains the classes that are mentioned in IUE SE 115 class.

# TABLE OF CONTENTS

	<u>Page</u>
<b>ABSTRACT .....</b>	<b>I</b>
<b>TABLE OF CONTENTS.....</b>	<b>II</b>
<b>LIST OF FIGURES.....</b>	<b>III</b>
<b>1. INTRODUCTION.....</b>	<b>1</b>
<b>2. DESIGN AND IMPLEMENTATION.....</b>	<b>2</b>
2.1. CLASSES.....	2
2.2. DESIGN CHOICES .....	6
2.3. ALGORITHM USED .....	6
<b>3. CHALLENGES AND SOLUTIONS .....</b>	<b>6</b>
3.1. FIRST CHALLENGE .....	6
3.2. SECOND CHALLENGE .....	7
3.3. THIRD CHALLENGE .....	7
<b>4. HOW PROGRAM WORKS.....</b>	<b>7</b>
4.1. INPUT FILE READING .....	7
4.2. STORING MAP DATA.....	7
4.3. FINDING THE SHORTEST PATH.....	8
4.4. WRITING THE OUTPUT FILE .....	8
4.5. OUTPUT RESULTS .....	8
<b>5. PROGRAM EXECUTION AND OUTPUT .....</b>	<b>9</b>
5.1. TERMINAL VIEW.....	9
5.2. INPUT FILE.....	9
5.3. OUTPUT FILE .....	10
<b>6. CONCLUSION .....</b>	<b>11</b>

## LIST OF FIGURES

	<b><u>Page</u></b>
Figure 1. City class.....	2
Figure 2. Route class. ....	3
Figure 3. CountryMap class. ....	3
Figure 4. CountryMap class, addCity and addRoute methods.....	4
Figure 5. CountryMap class, cityFind, starting ending city getter setter and index finder methods.....	5
Figure 6. WayFinder class.....	5

## 1. INTRODUCTION

This project will read a text file which the file provides the city size, city names, route size, routes and city that you currently are and the city that you want to go. Program will be initialized by only providing the path through terminal with the program execution. Program ensures input file is in the desired ordered if not it will give error where the problem occurred and why it occurs. Program will read the file and check if there any kind of error. If everything is in the correct order, program will find the shortest path using implementation of the idea that Dijkstra's algorithm have and write it into output.txt file inside the execution file or the file that .jar file executed.

This project is took 7-10 days to developed along with the 36 commits on GitHub.

In design and implementation choose inside the program have only the static array objects and the libraries that are mentioned in SE 115 class.

## 2. DESIGN AND IMPLEMENTATION

### 2.1. Classes

- **City:**

This is class stores city names with a unique label. Later this class will be used as an array to store city names in CountryMap class. Class can be seen in Figure 1.

```
1 public class City {  
2  
3     private String name;  
4  
5     public City(String name) {  
6         this.name = name;  
7     }  
8  
9     public String getName() {  
10        return name;  
11    }  
12  
13    public void setName(String name) {  
14        this.name = name;  
15    }  
16 }
```

Figure 1. City class.

- **Route:**

This class store routes from point a to b and distances. Like from city A to city B distance is 30 minutes. Also, City class is used as to hold city names in this class. So, in order to put city name, it must be City class object.

In Figure 2, getCity1, getCity2 and getDistance methods are used inside WayFinder class for finding the indexes of starting city and ending city.

Also getDistance method is used in WayFinder class to calculate the distance by summation.

```

1  public class Route { 4 usages
2      private City city1; 2 usages
3      private City city2; 2 usages
4      private int distance; 2 usages
5
6      public Route(City city1, City city2, int distance) { 1 usage
7          this.city1 = city1;
8          this.city2 = city2;
9          this.distance = distance;
10     }
11
12     public City getCity1() { 1 usage
13         return city1;
14     }
15
16     public City getCity2() { 1 usage
17         return city2;
18     }
19
20     public int getDistance() { 2 usages
21         return distance;
22     }
23 }

```

Figure 2. Route class.

- **CountryMap:**

This class stores the fixed size array of city and route class. In order to do that it initializes by providing the city and route size. It also stores the starting city and ending city later to be used on WayFinder class. Everything inside the text file will be written to this class. Class can be seen in Figure 3.

```

1  public class CountryMap { 6 usages
2
3      public City[] cities; 10 usages
4      public Route[] routes; 3 usages
5      private int citySize; 2 usages
6      private int routeSize; 2 usages
7
8      private String startCity; 2 usages
9      private String endCity; 2 usages
10
11     public CountryMap(int citySize, int routeSize) { 1 usage
12         this.cities = new City[citySize];
13         this.routes = new Route[routeSize];
14         this.citySize = citySize;
15         this.routeSize = routeSize;
16     }

```

Figure 3. CountryMap class.

A set of cityCount and routeCount is added to the code to make sure it doesn't exceed the size limits.

The addCity and addRoute methods are used to add routes to the CountryMap class. A counter ensures it doesn't exceed the size limits. Additionally, to ensure that the city names inside the routes are part of the added cities, an if statement checks if the city name is not included in the list of cities; if it is not, it is not written. Related methods can be seen in Figure 4.

```
1  public class CountryMap { 6 usages
18     private int cityCount= 0; 2 usages
19     private int routeCount= 0; 2 usages
20
21     public void addCity(String cityName){ 1 usage
22         if(cityCount < citySize){
23             cities[cityCount++] = new City(cityName);
24         }
25     }
26
27     public void addRoute(String from, String to, int dist){ 1 usage
28         if(cityFind(from) != null && cityFind(to) != null && routeCount < routeSize){
29             City fromCity = cityFind(from);
30             City toCity = cityFind(to);
31             routes[routeCount++] = new Route(fromCity, toCity, dist);
32         }
33     }
```

Figure 4. CountryMap class, addCity and addRoute methods.

In Figure 5, cityFind is to make sure that the related city name are in the cities array. If city name exist it returns the name of the city if not it returns null. This method is used inside the CountryMap in the addRoute method so that's why it's a private method.

Also inside Figure 5, cityIndexFind method is initialized by providing the city name that you want to find the index of. Then it looks cities array to where the index of that city. If it doesn't exist inside the cities array it returns -1.

The public String cityFind method is used in WayFinder class to writePath method. That returns the name of the city from the index.

The get, set methods for Start and End cities are to store the starting city and ending cities.

```

1  public class CountryMap { 6 usages
35 @ private City cityFind(String cityName){ 4 usages
36     for(City city : cities){
37         if(city.getName().equals(cityName)){
38             return city;
39         }
40     }
41     return null;
42 }
43
44 public int cityIndexFind(String cityName){ 6 usages
45     int i = 0;
46     for(City city : cities){
47         if(city.getName().equals(cityName)){
48             return i;
49         }
50         i++;
51     }
52     return -1;
53 }
54
55 > public String cityFind(int index) { return cities[index].getName(); }
58
59 > public String getStartCity() { return startCity; }
62
63 > public void setStartCity(String startCity) { this.startCity = startCity; }
66
67 > public String getEndCity() { return endCity; }
70
71 > public void setEndCity(String endCity) { this.endCity = endCity; }
74 }

```

Figure 5. CountryMap class, cityFind, starting ending city getter setter and index finder methods.

- **WayFinder:**

This class uses Dijkstra algorithm to find the shortest path in CountryMap. In Figure 6, it initialized by taking CountryMap object and using the Dijkstra algorithm (later to be explained in design part of this paper). Also, for writing the output into the file it has a method called writePath which takes the name of the place that you want to go and the starting point and returns string for later to be used on FileWrite class.

```

1  public class WayFinder { 2 usages
2
3     private CountryMap countryMap; 15 usages
4     private int[] distances; 17 usages
5     private boolean[] visited; 4 usages
6     private int[] previous; 8 usages
7
8     public WayFinder(CountryMap countryMap) { 1 usage
9         this.countryMap = countryMap;
10    }

```

Figure 6. WayFinder class.



- **FileRead:**

This class is for read the text file and give if there is an error in the file structured where the thing inside the file is not in the correct order it will give error descriptions as well as the which line the error occurs. It also catches if file does not exist. After it read city size, city names in a different String array and route size it begins writing the information into the CountryMap class and the routes as well.

- **FileWrite:**

This class is for if the shortest path exist this class will take writePath method inside the WayFinder class and write the wanted output into the output.txt file inside the src folder.

- **TestMain:**

This class is the main class which operates everything and inside this class we get the file name from terminal as 'args' and call other functions to read, store, find the shortest path and lastly write it into the output file.

## **2.2. Design Choices**

For clear code City and Route class created where Route class uses City objects to store route data. For CountryMap basic Arrays of City and Route classes used for storage and for the fix size of Array CountryMap will initialize by only providing the size of city and routes. To use the object-oriented programming more FileRead and FileWrite class created and later used for storage of CountryMap and creating of output.txt file. Lastly, to initialize everything and take the map file from terminal TestMain class used.

## **2.3. Algorithm Used**

To find the shortest path Dijkstra's algorithm used which the basic idea is taken from [geeksforgeeks.org](https://www.geeksforgeeks.org/). Since our path will be non-negative and all paths are vertical this method was the shortest and most efficient way to implement it. Otherwise, its algorithm will be brute force and static therefore will not work if it reaches more than the loop sized.

## **3. CHALLENGES AND SOLUTIONS**

### **3.1. First Challenge**

At first not using dynamic array like ArrayList or List for managing the city and route class inside the CountryMap was the main challenges that I come across with.

To solve this problem, I store the city size, city names and route size in a separate object inside FileRead and then initialize CountryMap by providing the city and route size then adding the city names accordingly. After that this problem were solved.

### **3.2. Second Challenge**

Second challenge was what will be the search algorithm. First, I try to do brute force by going one by one however size of the routes and cities can be change so this only work inside certain sizes and were not efficient way to do it. In my search for algorithms, I come across several of them and for my project the best algorithm was Dijkstra's algorithm. I take the general idea and base code from [geeksforgeeks.org](https://www.geeksforgeeks.org/) and change it according to my needs.

### **3.3. Third Challenge**

Third challenge was the error handling for every occasion. For example, one of the challenges was to when city size greater than the city name it can sometimes read the file and doesn't give any errors. To solve this problem, I put an error check mechanism where the program check if there are anything other than letter inside the city names.

Similar challenge to this were putting the where the errors occur. I solve this problem by putting the line numbers manually when it encounters the specific error (I check it with if else statements) until I came up with the loop then I made it automated (like when reading the route name and distance).

## **4. HOW PROGRAM WORKS**

### **4.1. Input File Reading**

First program starts with taking the input file name through the terminal along with the program execution. Then it starts read the file and make sure everything is in the desired order if not program will give the detailed description of the error and which line it occurs. Ensures that city name has only letters and file doesn't have anything left to read after we read accordingly to city size and route size. If file has more data that it should have it will give error.

In this part of the code program ensures that input file is read and not have a single error inside.

### **4.2. Storing Map Data**

While program reads the input file when it come to the line 3 (route size) it will start storing everything inside the file into the CountryMap class. Program also have error check mechanism inside CountryMap and Route classes as well incase city names and route names doesn't match, or program tries to store more data than the size restrictions.

An array of City objects stores the city names inside the CountryMap class.

An array of Route objects stores the route paths inside the CountryMap class and ensures that the city names and route names are match.

Start and end cities name are also stored inside the CountryMap class.

#### **4.3. Finding the Shortest Path**

WayFinder class that takes CountryMap as an input tries to find the shortest path exist implementing the Dijkstra's Algorithm of the program purpose. It tries to find the shortest path by setting the starting point distance 0 and everything else INF. Then check where the starting point connect other cities and how long it takes. It looks every way that starting point visit and the cities that along with that visited.

If path exist it, I change the printPath method to make sure it returns String and implement my method on writePath to returns the path in a desired output as a String. Later on, this will used as an input of FileWrite class as a content.

#### **4.4. Writing the Output File**

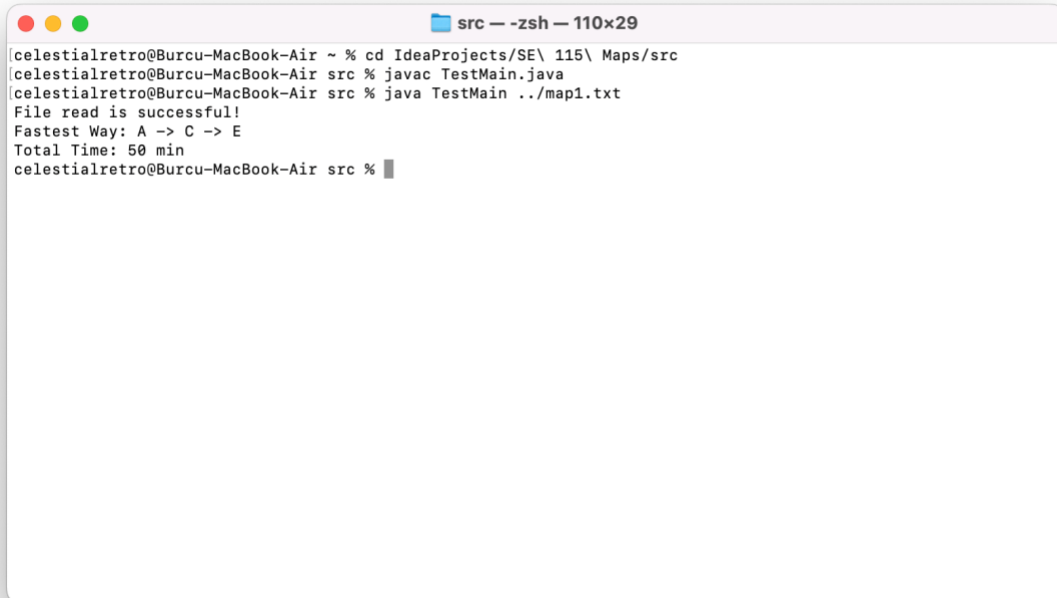
The shortest path and total travel time is calculated and ready to print with writePath method inside the WayFinder class. This method returns string to write the content to the writeToFile method inside the FileWrite class. This method will create new file called output.txt and write the output of the code.

#### **4.5. Output Results**

If the shortest path is found, the program displays the results on the console and writes them to the output file. If no valid path exists, the program informs the user.

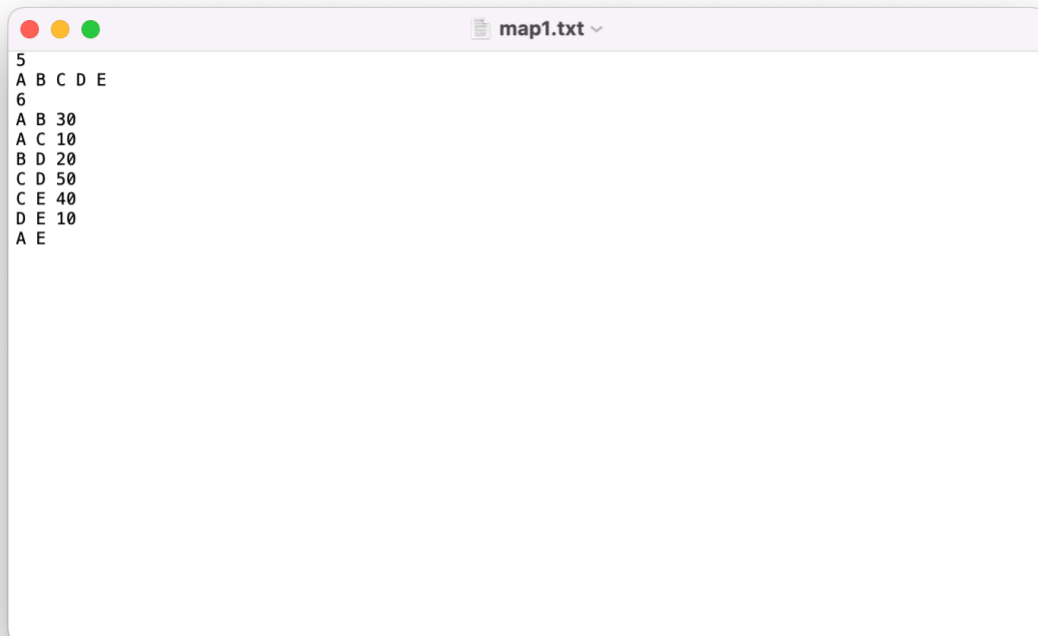
## 5. PROGRAM EXECUTION AND OUTPUT

### 5.1. Terminal View

A terminal window titled 'src — zsh — 110x29' showing the execution of a Java program. The user runs 'cd IdeaProjects/SE\ 115\ Maps/src', 'javac TestMain.java', and 'java TestMain ../map1.txt'. The output indicates a successful file read, the fastest path A->C->E, and a total time of 50 minutes.

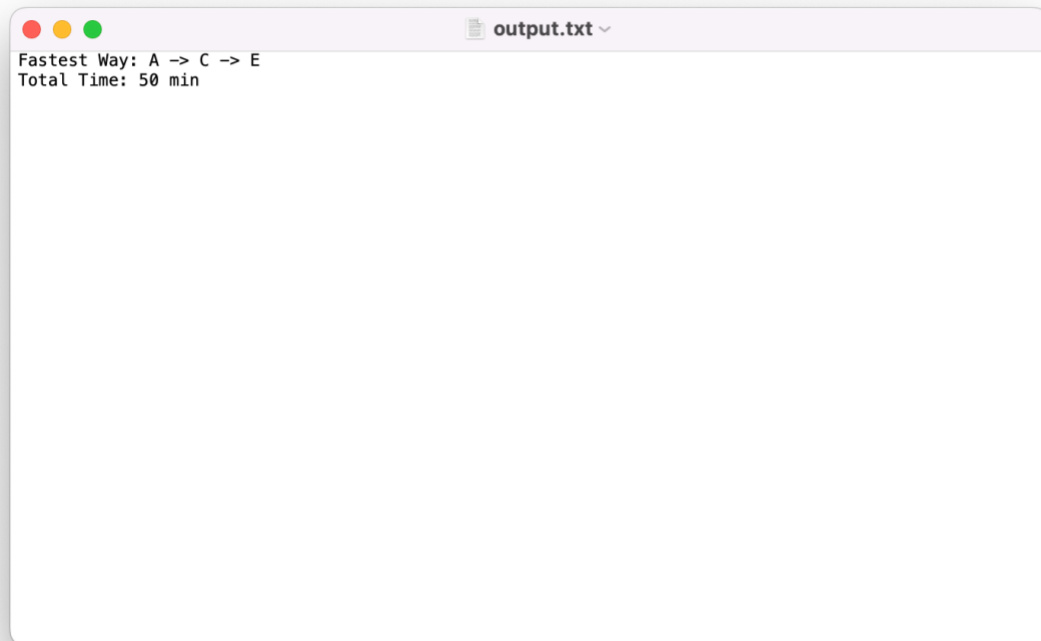
```
celestialretro@Burcu-MacBook-Air ~ % cd IdeaProjects/SE\ 115\ Maps/src  
celestialretro@Burcu-MacBook-Air src % javac TestMain.java  
celestialretro@Burcu-MacBook-Air src % java TestMain ../map1.txt  
File read is successful!  
Fastest Way: A -> C -> E  
Total Time: 50 min  
celestialretro@Burcu-MacBook-Air src %
```

### 5.2. Input File

A text editor window titled 'map1.txt' displaying the content of the input file. The file contains a graph structure with 5 nodes (A, B, C, D, E) and weighted edges between them.

```
5  
A B C D E  
6  
A B 30  
A C 10  
B D 20  
C D 50  
C E 40  
D E 10  
A E
```

### 5.3. Output File



## **6. CONCLUSION**

Code work as described in the functional requirements and all the nonfunctional requirements are met.